

UNITED STATES PATENT APPLICATION FOR

CONDITIONAL BRANCHING IN AN IN-CIRCUIT EMULATION SYSTEM

Inventors:

Craig Nemecek

Prepared by:

WAGNER, MURABITO & HAO, LLP

Two North Market Street

Third Floor

San Jose, California 95113

(408) 938-9060

1
2
3
4
5 **CONDITIONAL BRANCHING IN AN IN-CIRCUIT EMULATION SYSTEM**
6
7
8

9 **CROSS REFERENCE TO RELATED DOCUMENTS**

10 This application is a continuation-in-part of U.S. Patent Application Serial No.
11 09/975,105 filed October 10, 2001 to Nemecek entitled " Host to FPGA Interface
12 in an In-Circuit Emulation System", which is hereby incorporated. The application
13 is related to, incorporates by reference and claims priority benefit under 35 U.S.C.
14 §119(e) of provisional patent application serial no. 60/243,708 filed October 26,
15 2000 to Snyder, et al. entitled "Advanced Programmable Microcontroller Device"
16 which is also hereby incorporated herein by reference.
17

18 **COPYRIGHT NOTICE**

19 A portion of the disclosure of this patent document contains material which
20 is subject to copyright protection. The copyright owner has no objection to the
21 facsimile reproduction of the patent document or the patent disclosure, as it
22 appears in the Patent and Trademark Office patent file or records, but otherwise
23 reserves all copyright rights whatsoever.
24

25 **FIELD OF THE INVENTION**

26 This invention relates generally to the field of In-Circuit Emulation. More
27 particularly, this invention relates to methods and apparatus for providing
28 conditional branching in an In-Circuit Emulation system.
29

BACKGROUND OF THE INVENTION

In-circuit emulation (ICE) has been used by software and hardware developers for a number of years as a development tool to emulate the operation of complex circuit building blocks and permit diagnosis and debugging of hardware and software. Such in-circuit emulation is most commonly used currently to analyze and debug the behavior of complex devices such as microcontrollers and microprocessors that have internal structures that are far too complex to readily model using computer simulation software alone.

FIGURE 1 illustrates an exemplary conventional in-circuit emulation arrangement 100 used to model, analyze and debug the operation of a microcontroller device. In this arrangement, a host computer (e.g., a personal computer) 110 is connected to a debug logic block 120 which is further connected to a special version of the microcontroller device that has been developed specially for use in emulation. Operational instructions are loaded from the host computer 110 through the debug logic 120 to the special version of the microcontroller 130. The debug logic 120 monitors operation of the microcontroller 130 as the instructions are executed. Depending upon the application, this operation may be monitored while the special version of the microcontroller 130 is interconnected with the circuitry that is intended to interface a production version of the microcontroller in the finished product under development. Such interconnection may be via simulation within host computer 110 or as actual circuitry or some combination thereof. As the circuit is stepped through its operation, the debug logic gathers information about the state of various components of the microcontroller 130 during operation and feeds that information back to the host computer 110 for analysis.

During the course of the analysis, various trace information such as time stamps, register values, data memory content, etc. may be logged in the host computer 110 for analysis and debugging by the designer. Additionally, it is generally the case that various break points can be defined by the designer that

1 cause the program to halt execution at various points in the operation to permit
2 detailed analysis. Other debugging tools may also be provided to enable the user
3 to debug the operation of the circuit.

4 In-circuit emulation systems such as 100 have a number of disadvantages
5 and limitations. In earlier systems, the microcontroller 130 might have been simply
6 the production version of the microcontroller itself with no special debugging
7 features. In such systems, the information that can be gathered by the ICE system
8 100 is limited to that which is available at the pinouts of the microcontroller 130 (or
9 which can be extracted from the microcontroller using clever programming or
10 special coding supported by the processor).

11 Enhancements to such early systems provided the microcontroller or other
12 processor with an array of built-in debugging tools that use standard pins on the
13 part and built-in instructions that facilitated in-circuit emulation. In such enhanced
14 processors, the emulation tools are integrated into the part and thus become a
15 design constraint for developing and improving the part. Thus, support for the
16 debugging instruction code and the like can increase the cost and complexity of the
17 circuit.

18 Newer systems often use a so-called "bond-out" microcontroller. A bond-out
19 version of the microcontroller is a version of the production microcontroller that has
20 been designed with special wirebonding pads on the chip that are not normally
21 connected in the production wirebonding. The bond-out version connects these
22 pads to pins on the microcontroller package to permit access to otherwise
23 inaccessible points of the circuit to facilitate debugging. This technique is in
24 common use, but has the disadvantage of imposing significant limitations on the
25 circuit layout to permit space and circuitry associated with the special wirebonding
26 pads. Additionally, it is usually the case that substantial interface circuitry and
27 other special circuitry to facilitate the debugging and bond-out has to be added to
28 the circuit. This increases the complexity, size, power consumption and potentially
29 reduces the yield of the production part. Moreover, development resources are
30 required to lay out the bond-out circuitry and pads and do associated design of

1 such bond-out circuitry. Additionally, instruction code must generally be provided
2 and supported for such an implementation. Such resources may have to be
3 applied with every updated version of the part and may significantly impact speed,
4 cost or flexibility in development of improved versions of the part.

5 A third technique, one that is used in the Pentium™ and Pentium Pro™
6 series of microprocessors available from Intel Corporation, provides a special
7 "probe mode" of operation of the processor. When the processor is placed in this
8 probe mode, a number of internal signals are routed to a "debug port" for use by the
9 in-circuit emulation system. This debug port is used to permit the in-circuit
10 emulation system to communicate with the processors at all times and, when
11 placed in probe mode, to read otherwise inaccessible probe points within the
12 processor. Of course, providing such a probe mode requires significant design
13 resources to design in all such probe and debug functions and associated
14 instruction code support into the standard processor. This, of course, increases
15 development cost, chip complexity and chip size. Moreover, such facilities become
16 a part of the processor design which must be carried through and updated as
17 required as enhancements to the original design are developed.

18 19 SUMMARY OF THE INVENTION

20 The present invention relates generally to handling conditional jumps in an
21 ICE system. Objects, advantages and features of the invention will become
22 apparent to those skilled in the art upon consideration of the following detailed
23 description of the invention.

24 In one embodiment consistent with the present invention, an In-Circuit
25 Emulation system has a real microcontroller (device under test) operating in lock-
26 step with a virtual microcontroller so that registers, memory locations and other
27 debugged data can be retrieved from the virtual microcontroller without disrupting
28 operation of a real microcontroller. When an I/O read instruction is carried out
29 followed by a conditional jump instruction dependent upon the I/O read data, the
30 virtual microcontroller does not have adequate time to compute the jump address

1 after receipt of I/O read data from the real microcontroller. Thus, when this
2 sequence of instructions is detected, the virtual microcontroller pre-calculates the
3 jump address and makes the jump decision after receipt of the I/O read data from
4 the real microcontroller. This is advantageous in that it permits the virtual
5 microcontroller to operate in lock-step with the real microcontroller so that debug
6 information can be retrieved from the virtual microcontroller to avoid disturbing
7 operation of the real microcontroller and permit real time debugging.

8 An in-circuit emulation system consistent with an embodiment of the present
9 invention has a microcontroller. A virtual microcontroller is coupled to and executes
10 instructions in lock-step with the microcontroller, The microcontroller sends I/O read
11 data to the virtual microcontroller. The virtual microcontroller detects a sequence
12 of instructions comprising an I/O read instruction followed by a conditional jump
13 instruction, and for computing a conditional jump address prior to receipt of I/O read
14 data from the microcontroller. The virtual microcontroller further determines after
15 receipt of the I/O read data from the microcontroller whether to proceed with
16 instruction execution at a next consecutive address or at the conditional jump
17 address.

18 In an in-circuit emulation system having a microcontroller coupled to and
19 operating in lock-step with a virtual microcontroller, a method consistent with
20 certain embodiments of the present invention for handling conditional jumps in the
21 virtual microcontroller, includes detecting a sequence of instructions comprising an
22 I/O read instruction followed by a conditional jump instruction; computing a
23 conditional jump address prior to receipt of I/O read data from the microcontroller;
24 and determining after receipt of the I/O read data from the microcontroller whether
25 to proceed with instruction execution at a next consecutive address or at the
26 conditional jump address.

27 In another in-circuit emulation system having a device under test coupled to
28 and operating in lock-step with a virtual processor, a method consistent with certain
29 embodiments of the present invention, of handling conditional jumps in the virtual
30 processor, includes detecting a sequence of instructions comprising an I/O read

1 instruction followed by a conditional jump instruction; computing a conditional jump
2 address prior to receipt of I/O read data from the virtual processor; and determining
3 after receipt of the I/O read data from the device under test whether to proceed with
4 instruction execution at a next consecutive address or at the conditional jump
5 address.

6 The above summaries are intended to illustrate exemplary embodiments of
7 the invention, which will be best understood in conjunction with the detailed
8 description to follow, and are not intended to limit the scope of the appended
9 claims.

10 11 BRIEF DESCRIPTION OF THE DRAWINGS

12 The features of the invention believed to be novel are set forth with
13 particularity in the appended claims. The invention itself however, both as to
14 organization and method of operation, together with objects and advantages
15 thereof, may be best understood by reference to the following detailed description
16 of the invention, which describes certain exemplary embodiments of the invention,
17 taken in conjunction with the accompanying drawings in which:

18 **FIGURE 1** is a block diagram of a conventional In-Circuit Emulation system.

19 **FIGURE 2** is a block diagram of an exemplary In-Circuit Emulation system
20 consistent with certain microcontroller embodiments of the present invention.

21 **FIGURE 3** is an illustration of the operational phases of an In-Circuit
22 Emulation system consistent with an embodiment of the present invention.

23 **FIGURE 4** is an illustration of the operational phases of an In-Circuit
24 Emulation system consistent with an embodiment of the present invention viewed
25 from a virtual microcontroller perspective.

26 **FIGURE 5** is a timing diagram useful in understanding an exemplary data
27 and control phase of operation of certain embodiments of the present invention.

28 **FIGURE 6** is a block diagram isolating the host to FPGA interface consistent
29 with an embodiment of the present invention

1 **FIGURE 7** is a flow chart describing the operation of the host to FPGA
2 interface in an embodiment consistent with the present invention.

3 **FIGURE 8** is a flow chart describing conditional branching consistent with
4 an embodiment of the present invention.

6 **DETAILED DESCRIPTION OF THE INVENTION**

7 In the following detailed description of the present invention, numerous
8 specific details are set forth in order to provide a thorough understanding of the
9 present invention. However, it will be recognized by one skilled in the art that the
10 present invention may be practiced without these specific details or with
11 equivalents thereof. In other instances, well known methods, procedures,
12 components, and circuits have not been described in detail as not to unnecessarily
13 obscure aspects of the present invention.

15 **NOTATION AND NOMENCLATURE**

16 Some portions of the detailed descriptions which follow are presented in
17 terms of procedures, steps, logic blocks, processing, and other symbolic
18 representations of operations on data bits that can be performed on computer
19 memory. These descriptions and representations are the means used by those
20 skilled in the data processing arts to most effectively convey the substance of their
21 work to others skilled in the art. A procedure, computer executed step, logic block,
22 process, etc., is here, and generally, conceived to be a self-consistent sequence
23 of steps or instructions leading to a desired result. The steps are those requiring
24 physical manipulations of physical quantities.

25 Usually, though not necessarily, these quantities take the form of electrical
26 or magnetic signals capable of being stored, transferred, combined, compared, and
27 otherwise manipulated in a computer system. It has proven convenient at times,
28 principally for reasons of common usage, to refer to these signals as bits, values,
29 elements, symbols, characters, terms, numbers, or the like.

1 It should be borne in mind, however, that all of these and similar terms are
2 to be associated with the appropriate physical quantities and are merely convenient
3 labels applied to these quantities. Unless specifically stated otherwise as apparent
4 from the following discussions, it is appreciated that throughout the present
5 invention, discussions utilizing terms such as "processing" or "transferring" or
6 "executing" or "determining" or "instructing" or "issuing" or "halting" or "clearing" or
7 the like, refer to the action and processes of a computer system, or similar
8 electronic computing device, that manipulates and transforms data represented as
9 physical (electronic) quantities within the computer system's registers and
10 memories into other data similarly represented as physical quantities within the
11 computer system memories or registers or other such information storage,
12 transmission or display devices.

13
14 **CONDITIONAL BRANCHING IN AN IN-CIRCUIT EMULATION SYSTEM IN**
15 **ACCORDANCE WITH THE INVENTION**

16 While this invention is susceptible of embodiment in many different forms,
17 there is shown in the drawings and will herein be described in detail specific
18 embodiments, with the understanding that the present disclosure is to be
19 considered as an example of the principles of the invention and not intended to limit
20 the invention to the specific embodiments shown and described. In the description
21 below, like reference numerals are used to describe the same, similar or
22 corresponding parts in the several views of the drawings.

23 A commercial ICE system utilizing the present invention is available from
24 Cypress Micro Systems, Inc., for the CY8C25xxx/26xxx series of microcontrollers.
25 Detailed information regarding this commercial product is available from Cypress
26 Micro Systems, Inc., 22027 17th Avenue SE, Suite 201, Bothell, WA 98021Bothell,
27 WA in the form of version 1.11 of "PSoC Designer: Integrated Development
28 Environment User Guide", which is hereby incorporated by reference. While the
29 present invention is described in terms of an ICE system for the above exemplary

1 microcontroller device, the invention is equally applicable to other complex circuitry
2 including microprocessors and other circuitry that is suitable for analysis and
3 debugging using in-circuit emulation. Moreover, the invention is not limited to the
4 exact implementation details of the exemplary embodiment used herein for
5 illustrative purposes.

6 Referring now to **FIGURE 2**, an architecture for implementation of an
7 embodiment of an ICE system of the present invention is illustrated as system 200.
8 In system 200, a Host computer 210 (e.g., a personal computer based on a
9 Pentium™ class microprocessor) is interconnected (e.g., using a standard PC
10 interface 214 such as a parallel printer port connection, a universal serial port
11 (USB) connection, etc.) with a base station 218. The host computer 210 generally
12 operates to run an ICE computer program to control the emulation process and
13 further operates in the capacity of a logic analyzer to permit a user to view
14 information provided from the base station 218 for use in analyzing and debugging
15 a system under test or development.

16 The base station 218 is based upon a general purpose programmable
17 hardware device such as a gate array configured to function as a functionally
18 equivalent "virtual microcontroller" 220 (or other device under test (DUT)). This is
19 accomplished using an associated integral memory 222 which stores program
20 instructions, data, trace information and other associated information. Thus, the
21 base station is configured as an emulator of the internal microprocessor portion of
22 the microcontroller 232. In preferred embodiments, a field programmable gate
23 array FPGA (or other programmable logic device) is configured to function as the
24 virtual microcontroller 220. The FPGA and virtual microcontroller 220 will be
25 referred to interchangeably herein. The base station 218 is coupled (e.g., using a
26 four wire interface 226) to a standard production microcontroller 232 mounted in a
27 mounting device referred to as a "pod". The pod, in certain embodiments, provides
28 connections to the microcontroller 232 that permit external probing as well as
29 interconnection with other circuitry as might be used to simulate a system under
30 development.

1 The FPGA of the base station 218 of the current embodiment is designed
2 to emulate the core processor functionality (microprocessor functions, Arithmetic
3 Logic Unit functions and RAM and ROM memory functions) of the Cypress
4 CY8C25xxx/26xxx series microcontrollers. The CY8C25xxx/26xxx series of
5 microcontrollers also incorporates I/O functions and an interrupt controller as well
6 as programmable digital and analog circuitry. This circuitry need not be modeled
7 using the FPGA 220. Instead, the I/O read information, interrupt vectors and other
8 information can be passed to the FPGA 220 from the microcontroller 232 over the
9 interface 226 as will be described later.

10 In order to minimize the need for any special ICE related functions on the
11 microcontroller 232 itself, the FPGA 220 and associated circuitry of the base station
12 218 are designed to operate functionally in a manner identically to that of
13 microprocessor portion of the production microcontroller, but to provide for access
14 to extensive debug tools including readout of registers and memory locations to
15 facilitate traces and other debugging operations.

16 The base station 218's virtual microcontroller 220 operates to execute the
17 code programmed into the microcontroller 232 in lock-step operation with the
18 microcontroller 232. Thus, the actual microcontroller 232 is freed of any need to
19 provide significant special facilities for ICE, since any such facilities can be
20 provided in the virtual microcontroller 220. The base station 218's virtual
21 microcontroller 220 and microcontroller 232 operate together such that I/O reads
22 and interrupts are fully supported in real time. The combination of real and virtual
23 microcontroller behave just as the microcontroller 232 would alone under normal
24 operating conditions. I/O reads and interrupt vectors are transferred from the
25 microcontroller 232 to the base station 218 as will be described later. Base station
26 218 is then able to provide the host computer 210 with the I/O reads and interrupt
27 vectors as well as an array of information internal to the microcontroller 232 within
28 memory and register locations that are otherwise inaccessible.

29 In the designing of a microcontroller other complex circuit such as the
30 microcontroller 232, it is common to implement the design using the Verilog™

1 language (or other suitable language). Thus, it is common that the full functional
2 design description of the microcontroller is fully available in a software format. The
3 base station 218 of the current embodiment is based upon the commercially
4 available Spartan™ series of FPGAs from Xilinx, Inc., 2100 Logic Drive, San Jose,
5 CA 95124. The Verilog™ description can be used as the input to the FPGA design
6 and synthesis tools available from the FPGA manufacturer to realize the virtual
7 microcontroller 220 (generally after timing adjustments and other debugging).
8 Thus, design and realization of the FPGA implementation of an emulator for the
9 microcontroller (virtual microcontroller) or other device can be readily achieved by
10 use of the Verilog™ description along with circuitry to provide interfacing to the base
11 station and the device under test (DUT).

12 In the embodiment described in connection with **FIGURE 2**, the actual
13 production microcontroller 232 carries out its normal functions in the intended
14 application and passes I/O information and other information needed for debugging
15 to the FPGA 220. The virtual microcontroller 220 implemented within the FPGA of
16 base station 218 serves to provide the operator with visibility into the core processor
17 functions that are inaccessible in the production microcontroller 232. Thus, the
18 FPGA 220, by virtue of operating in lock-step operation with the microcontroller 232
19 provides an exact duplicate of internal registers, memory contents, interrupt vectors
20 and other useful debug information. Additionally, memory 222 can be used to store
21 information useful in trace operations that is gathered by the FPGA 220 during
22 execution of the program under test. This architecture, therefore, permits the
23 operator to have visibility into the inner workings of the microcontroller 232 without
24 need to provide special bondouts and expensive circuitry on the microcontroller
25 itself.

26 The base station 218's FPGA based virtual microcontroller 220, operating
27 under control of host computer 210, carries out the core processor functions of
28 microcontroller 232 and thus contains a functionally exact emulated copy of the
29 contents of the registers and memory of the real microcontroller 232. The ICE
30 system starts both microcontrollers (real and virtual) at the same time and keeps

them running in synchronization. The real microcontroller 232 sends I/O data to the base station 218 (and in turn to the ICE software operating on the host computer 210 if required) fast enough to keep the real microcontroller 232 and the virtual microcontroller 220 of base station 218 in synchronization. Whenever the system is halted (i.e., when the system is not emulating), other information such as flash memory programming functions, test functions, etc. can be sent over the interface.

Because the microcontroller 232 operates in synchronization with the virtual microcontroller 220, less data needs to be sent over the four wire interface than would be required in an ICE system otherwise. The type of data sent over the lines is allowed to change depending on when the data is sent in the execution sequence. In other words, depending on the execution sequence time, the information over the data lines can be commands to the real microcontroller 232 or they can be data. Since the clock frequency of the real microcontroller 232 is programmable, it copies its current clock on one of the lines of the four wire interface. Moreover, the lock-step operation of the microcontroller 232 and the virtual microcontroller 220 allows the virtual microcontroller 220 to not require certain resources of the microcontroller 232 such as timers, counters, amplifiers, etc. since they are fully implemented in the microcontroller 232. In addition, the microcontroller 232 (or other DUT) can be debugged in real time without need for extensive debug logic residing on the microcontroller 232, since all registers and memory locations, etc. are available through the virtual microcontroller 220.

In the embodiment illustrated, the basic interface used is a four line interface between microcontroller 232 and base station 218. This interface permits use of a standard five wire Category Five patch cable to connect the microcontroller 232 and base station 218 in one embodiment, but of course, this is not to be considered limiting. The four wire interface 226 of the present embodiment can be functionally divided into two functional portions. A data transport portion 242 carries two data lines in the current embodiment. A clock portion 246 carries a debug system clock plus the microcontroller clock signal for the microcontroller 232. Three additional

1 lines are also provided (not shown) for supply, ground and a reset line. But, the
2 data transport portion 242 and the clock portion 246 are of primary interest, since
3 the supply and reset functions can be readily provided in any other suitable manner.

4 The two portions of the interface are implemented in the current embodiment
5 using four lines as described, however, in other embodiments, these two portions
6 can be implemented with as few as two wires. In the current embodiment, the
7 microcontroller clock signal can be varied by programming (even dynamically
8 during execution of a program). Therefore, it is desirable to have two clock signals
9 - the microcontroller clock to easily track the microcontroller clock timing as well
10 as a system clock that regulates the data transfer and other operations. However,
11 in other embodiments, particularly where a clock frequency is not changed
12 dynamically, a single clock can be used. The single clock can be multiplied or
13 divided as required to implement the required clocking signals.

14 The present embodiment using an eight bit microcontroller that only reads
15 eight bits at a time on any given I/O read. Thus, the present microcontroller 232
16 needs only to effect serializing and transferring a maximum of one eight bit I/O read
17 for each instruction cycle. This is easily accommodated using two data lines
18 transferring four bits each over four system clock cycles. However, using a clock
19 which is two times faster, a single line could equally well transfer the data in the
20 same time. Similarly, four lines could be used to transfer the same data in only two
21 clock cycles. In any case, the objective is to transfer the data in a short enough
22 time to permit the virtual microcontroller 220 to process the data and issue any
23 needed response before the next instruction cycle begins. The time required to
24 accomplish this is held at a minimum in the current invention, since the system
25 synchronization eliminates need for any overhead protocol for transmission of the
26 data.

27 The current embodiment of the invention uses a four line communication
28 interface and method of communicating between the FPGA within base station 218
29 (acting as a "virtual microcontroller" 220 or ICE) and the real microcontroller device
30 under test (microcontroller 232). The four line communication interface is time-

dependent so that different information can be transferred at different times over a small number of communication lines. Moreover, since the two processors operate in lockstep, there is no need to provide bus arbitration, framing, or other protocol overhead to effect the communication between the microcontroller 232 and the virtual microcontroller 220. This interface is used for, among other things, transferring of I/O data from the microcontroller 232 to the FPGA 220 (since the FPGA emulates only the core processor functions of the microcontroller in the current embodiment). A first interface line (Data1) is a data line used by the microcontroller 232 to send I/O data to the FPGA based virtual microcontroller 220. This line is also used to notify the FPGA 220 of pending interrupts. This Data1 line is only driven by the real microcontroller 232. A second data line (Data2), which is bidirectional, is used by the microcontroller 232 to send I/O data to the FPGA based virtual microcontroller of base station 218. In addition, the FPGA 220 uses the Data2 line to convey halt requests (i.e., to implement simple or complex breakpoints) to the microcontroller 232.

A third interface line is a 24/48 Mhz debug system clock used to drive the virtual microcontroller 220's communication state machines (the logic used within the state controller to communicate with the microcontroller 232). In the current embodiment, this clock always runs at 24 MHz unless the microcontroller 232's internal clock is running at 24 Mhz. In this case the system clock switches to 48 Mhz. Of course, these exact clock speeds are not to be considered limiting, but are presented as illustrative of the current exemplary embodiment. The fourth interface line is the internal microcontroller clock from the microcontroller 232.

A fifth line can be used to provide a system reset signal to effect the simultaneous startup of both microcontrollers. This fifth line provides a convenient mechanism to reset the microcontrollers, but in most environments, the simultaneous startup can also be effected in other ways including switching of power. Sixth and Seventh lines are provided in the current interface to provide power and ground for power supply.

1 The base station 218's virtual microcontroller 220 communicates with the
2 microcontroller 232 via four signal and clock lines forming a part of the four line
3 interface 226 forming a part of a seven wire connection as described below. The
4 interface signals travel over a short (e.g., one foot) of CAT5 network cable. The ICE
5 transmits break commands to the microcontroller 232 via the base station 218,
6 along with register read/write commands when the microcontroller 232 is halted.
7 The microcontroller 232 uses the interface to return register information when
8 halted, and to send I/O read, interrupt vector, and watchdog information while
9 running. The microcontroller 232 also sends a copy of its internal clocks for the
10 ICE. The four lines of the four line interface are the first four entries in the table
11 below. Each of the signals and their purpose is tabulated below in **TABLE 1**:
12
13
14

Signal Name	Signal Direction with Respect to Base Station 218	Description
U_HCLK (Data Clock or HCLOCK)	In	24/48MHz data clock driven by microcontroller 232. This clock is used to drive the ICE virtual microcontroller communication state machines. This clock always runs at 24MHz, unless the U_CCLK clock is running at 24MHz — then it switches to 48MHz.
U_CCLK (microcontroller Clock or CCLOCK)	In	The internal microcontroller 232 CPU clock.
U_D1_IRQ (Data1)	In	One of two data lines used by the microcontroller 232 to send I/O data to the ICE. This line is also used to notify the ICE of pending interrupts. This line is only driven by the microcontroller 232 (i.e., unidirectional).
U_D0_BRQ (Data1)	In/Out	One of two data lines used by the microcontroller 232 to send I/O data to the ICE. The ICE uses this line to convey halt requests and other information to the microcontroller 232. This line is used for bi-directional communication.
ICE_POD_RST (RESET)	Out	Optional active high reset signal to microcontroller 232.
ICE_POD_PW_R (POWER)	Out	Optional power supply to microcontroller 232.
ICE_POD_GND (GROUND)	Out	Optional ground wire to microcontroller 232.

TABLE 1

1
2 Synchronization between the microcontroller 232 and the virtual
3 microcontroller 220 is achieved by virtue of their virtually identical operation. They
4 are both started simultaneously by a power on or reset signal. They then track
5 each other's operation continuously executing the same instructions using the
6 same clocking signals. The system clock signal and the microcontroller clock
7 signal are shared between the two microcontrollers (real and virtual) so that even
8 if the microprocessor clock is changed during operation, they remain in lock-step.

9 In accordance with certain embodiments of the invention, a mechanism is
10 provided for allowing the FPGA 220 of base station 218 and the microcontroller 232
11 to stop at the same instruction in response to a breakpoint event (a break or halt).
12 The FPGA 220 has the ability monitor the microcontroller states of microcontroller
13 232 for a breakpoint event, due to its lock-step operation with microcontroller 232.
14 In the process of executing an instruction, an internal start of instruction cycle (SOI)
15 signal is generated (by both microcontrollers) that indicates that the device is about
16 to execute a next instruction. If a breakpoint signal (a halt or break signal - the
17 terms "halt" and "break" are used synonymously herein) is generated by the FPGA,
18 the execution of the microcontroller 232 can be stopped at the SOI signal point
19 before the next instruction starts.

20 Although the SOI signal is labeled as a signal indicating the start of an
21 instruction, the SOI signal is used for multiple purposes in the present
22 microcontroller. It is not required that the SOI signal actually indicate a start of
23 instruction for many purposes, merely that there be a convenient time reference on
24 which to base certain actions. For example, any reference signal that always takes
25 place prior to execution of an instruction can be used as a time reference for
26 reading a halt command. Accordingly, any such available or generated reference
27 signal can be used equivalently as a "halt read" signal without departing from the
28 present invention. That notwithstanding, the SOI signal is conveniently used in the
29 current embodiment and will be used as a basis for the explanation that follows, but
30 should not be considered limiting.

1 Logic within the FPGA 220 of base station 218 allows not only for
2 implementation of simple breakpoint events, but also for producing breakpoints as
3 a result of very complex events. By way of example, and not limitation, a
4 breakpoint can be programmed to occur when a program counter reaches 0x0030,
5 an I/O write is happening and the stack pointer is about to overflow. Other such
6 complex breakpoints can readily be programmed to assist in the process of
7 debugging. Complex breakpoints are allowed, in part, also because the virtual
8 microcontroller 220 has time to carry out complex computations and comparisons
9 after receipt of I/O data transfers from the microcontroller 232 and before the next
10 instruction commences. After the receipt of I/O data from the microcontroller 232,
11 the FPGA 220 of base station 218 has a relatively long amount of computation time
12 to determine if a breakpoint event has occurred or not. In the event a breakpoint
13 has occurred, the microcontroller 232 can be halted and the host processor 210 is
14 informed.

15 An advantage of this process is that the FPGA 220 and the microcontroller
16 232 can be stopped at the same time in response to a breakpoint event. Another
17 advantage is that complex and robust breakpoint events are allowed while still
18 maintaining breakpoint synchronization between the two devices. These
19 advantages are achieved with minimal specialized debugging logic (to send I/O
20 data over the interface) and without special bond-out circuitry being required in the
21 microcontroller device under test 232.

22 Normal operation of the current microcontroller is carried out in a cycle of
23 two distinct stages or phases as illustrated in connection with **FIGURE 3**. The
24 cycle begins with the initial startup or reset of both the microcontroller 232 and the
25 virtual microcontroller 220 at 304. Once both microcontrollers are started in
26 synchronism, the data phase 310 is entered in which serialized data is sent from
27 the microcontroller to the virtual microcontroller. At the start of this phase the
28 internal start of instruction (SOI) signal signifies the beginning of this phase will
29 commence with the next low to high transition of the system clock. In the current
30 embodiment, this data phase lasts four system clock cycles, but this is only

1 intended to be exemplary and not limiting. The SOI signal further indicates that any
2 I/O data read on the previous instruction is now latched into a register and can be
3 serialized and transmitted to the virtual microcontroller. Upon the start of the data
4 phase 310, any such I/O read data (eight bits of data in the current embodiment)
5 is serialized into two four bit nibbles that are transmitted using the Data0 and Data1
6 lines of the current interface data portion 242. One bit is transmitted per data line
7 at the clock rate of the system clock. Thus, all eight bits are transmitted in the four
8 clock cycles of the data transfer phase.

9 At the end of the four clock cycle data transfer phase in the current
10 embodiment, the control phase 318 begins. During this control phase, which in the
11 current embodiment may be as short as two microcontroller clock periods (or as
12 long as about fourteen clock periods, depending upon the number of cycles
13 required to execute an instruction), the microcontroller 232 can send interrupt
14 requests, interrupt data, and watchdog requests. Additionally, the virtual
15 microcontroller 220 can issue halt (break) commands. If a halt command is issued,
16 it is read by the microcontroller at the next SOI signal. Once the control phase
17 ends, the data transfer phase repeats. If there is no data to transfer, data1 and
18 data2 remain idle (e.g., at a logic low state). To simplify the circuitry, I/O bus data
19 are sent across the interface on every instruction, even if it is not a bus transfer.
20 Since the virtual microcontroller 220 is operating in synchronization with
21 microcontroller 232 and executing the same instructions, the emulation system
22 knows that data transferred during non I/O read transfers can be ignored.

23 **FIGURE 4** shows this operational cycle from the perspective of the virtual
24 microcontroller 220. During the data transfer phase 310, the serialized data is
25 received over Data0 and Data1. It should be noted that prior to receipt of this I/O
26 data, the microcontroller 232 has already had access to this data for several clock
27 cycles and has already taken action on the data. However, until receipt of the I/O
28 read data during the data transfer phase 310, the virtual microcontroller 220 has not
29 had access to the data. Thus, upon receipt of the I/O read data during the data
30 phase 310, the virtual microcontroller 220 begins processing the data to catch up

1 with the existing state of microcontroller 232. Moreover, once the I/O data has been
2 read, the host computer 210 or virtual microcontroller 220 may determine that a
3 complex or simple breakpoint has been reached and thus need to issue a break
4 request. Thus, the virtual microcontroller should be able to process the data quickly
5 enough to make such determinations and issue a break request prior to the next
6 SOI. Break requests are read at the internal SOI signal, which also serves as a
7 convenient reference time marker that indicates that I/O data has been read and
8 is available for transmission by the microcontroller 232 to the virtual microcontroller
9 220.

10 By operating in the manner described, any breakpoints can be guaranteed
11 to occur in a manner such that both the virtual microcontroller 220 and the
12 microcontroller 232 halt operation in an identical state. Moreover, although the
13 virtual microcontroller 220 and the microcontroller 232 operate on I/O data obtained
14 at different times, both microcontrollers are in complete synchronization by the time
15 each SOI signal occurs. Thus, the virtual microcontroller 220 and the
16 microcontroller 232 can be said to operate in lock-step with respect to a common
17 time reference of the SOI signal as well as with respect to execution of any
18 particular instruction within a set of instructions being executed by both virtual
19 microcontroller 220 and the microcontroller 232.

20 A transfer of I/O data as just described is illustrated with reference to the
21 timing diagram of **FIGURE 5**. After the microcontroller 232 completes an I/O read
22 instruction, it sends the read data back to the base station 218 to the virtual
23 microcontroller, since the virtual microcontroller 220 of the present embodiment
24 implements only the core processor functions (and not the I/O functions). The ICE
25 system can expect the incoming data stream for an I/O read to commence with the
26 first positive edge of U_HCLK (the debug or system clock) when SOI signal for the
27 following instruction is at a predetermined logic level (e.g., a logic high). Thus, at
28 time T1, the SOI signal makes a transition to a logic high and one system clock
29 cycle later at time T2, the data transfer phase 310 begins. This timing allows the
30 ICE system to get the read data to the emulated accumulator of base station 218

1 before it is needed by the next instruction's execution. Note that the first SOI pulse
2 shown in **FIGURE 5** represents the first SOI following the I/O read instruction (but
3 could be any suitable reference time signal). Transfer of the data from the
4 microcontroller 232 is carried out using the two data lines (data2 and data1, shown
5 as U_D0_BRK and U_D1_IRQ) with each line carrying four bits of an eight bit word.
6 During this data transfer phase 310, an eight bit transfer representing the I/O read
7 data can take place from the microcontroller 232 to the base station 210 in the four
8 clock cycles between T2 and T3. The control phase 318 starts at time T3 and
9 continues until the beginning of the next data transfer phase 310. The SOI signal
10 at T4 indicates that the next data transfer phase is about to start and serves as a
11 reference time to read the data2 line to detect the presence of any halt signal from
12 the virtual microcontroller 220. The current control phase 318 ends at T5 and the
13 next data transfer phase 310 begins.

14 The base station 218 only transmits break (halt) commands to the
15 microcontroller 232 during the control phase. After the microcontroller 232 is halted
16 in response to the break command, the interface can be used to implement
17 memory / register read / write commands. The halt command is read at the SOI
18 signal transition (T1 or T4). The microcontroller 232 uses the interface to return
19 register information when halted, and to send I/O read, interrupt vector and
20 watchdog timer information while running.

21 To summarize, a break is handled as follows: The ICE asserts U_D0_BRQ
22 (break) to stop the microcontroller 232. When the ICE asserts the break, the
23 microcontroller 232 reads it at the SOI transition to high and stops. The ICE assert
24 breaks during the control phase. The microcontroller 232 samples the U_D0_BRQ
25 line at the rising edge of SOI (at T4) to determine if a break is to take place. After
26 halting, the ICE may issue commands over the U_D0_BRQ line to query the status
27 of various registers and memory locations of the virtual microcontroller or carry out
28 other functions.

29 In the case of an interrupt, if an interrupt request is pending for the
30 microcontroller 232, the system asserts U_D1_IRQ as an interrupt request during

1 the control phase of the microcontroller 232. Since the interrupt signal comes to
2 the virtual microcontroller 220 from the microcontroller 232 during the control
3 phase, the virtual microcontroller 220 knows the timing of the interrupt signal going
4 forward. That is, the interrupt signal is the synchronizing event rather than the SOI
5 signal. In case of an interrupt, there is no SOI, because the microcontroller 232
6 performs special interrupt processing including reading the current interrupt vector
7 from the interrupt controller. Since program instructions are not being executed
8 during the interrupt processing, there is no data / control phase. The virtual
9 microcontroller 220 expects the interrupt vector to be passed at a deterministic time
10 across the interface during this special interrupt processing and before execution
11 of instructions proceeds. Since the virtual microcontroller 220 of the current
12 embodiment does not implement an interrupt controller, interrupt vectors are read
13 from the interrupt controller upon receipt of an interrupt request over the interface.
14 The interrupt vector data is passed over the interface using the two data lines as
15 with the I/O read data, following the assertion of an internal microcontroller IVR_N
16 (active low) signal during the control phase. In the current embodiment, an
17 interrupt cycle is approximately 10 clock cycles long. Since the interrupt service
18 cycle is much longer than the time required to transfer the current interrupt vector,
19 the data is easily transferred using the two data lines, with no particular timing
20 issues.

21 If the microcontroller 232 undergoes a watchdog reset, it asserts the IRQ
22 (interrupt) and BRQ (break) lines indefinitely. The ICE detects this condition and
23 further detects that the microcontroller clock has stopped. This is enough to
24 establish that a watchdog reset has occurred. The ICE applies an external reset,
25 and notifies the ICE software in the host computer 210.

26 Referring now to the block diagram of **FIGURE 6**, the interface between the
27 host processor 210 and the base station 218 of a preferred embodiment of the
28 present invention is illustrated. In this embodiment, the connection between the
29 host processor 210 and the FPGA 220 is advantageously provided using a standard
30 IEEE 1284 parallel printer cable 214 with communication carried out using a

1 modification of standard EPP (enhanced parallel port) communication protocol. Of
2 particular interest in this communication interface is the data strobe connection
3 412, the INIT (initialize) connection 416 and the eight data connections (data line
4 0 through data line 7) 420. These connection are directly connected to the FPGA
5 with the INIT connection connected to the FPGA RESET pin. The data strobe line
6 412 is connected to the FPGA configuration clock input and the eight data lines 420
7 are connected to data input pins of the FPGA.

8 When the software on the host is started, the INIT connection 416 is driven
9 by the host computer 210 to a logic low causing the FPGA to clear its configuration
10 memory 424 and begin receiving configuration data. The configuration data is
11 stored in configuration memory to define the functionality of the FPGA. This
12 configuration data is clocked in eight bits at a time over the data lines 420 using the
13 data strobe signal as a clock signal. That is, an eight bit word is placed on the
14 interface data lines 420 by host processor 210 followed by toggling the data strobe
15 line to clock the data into the FPGA 220. This unidirectional data transfer from the
16 host computer incorporates a set of design parameters that configure the circuitry
17 of the FPGA 220 to function, in part, as a standard IEEE 1284 EPP interface once
18 the FPGA 220 is programmed and functional. This programming configures the
19 FPGA 220 to have an IEEE 1284 EPP interface with the data lines 420 connected
20 to the FPGA as bidirectional data lines, the configuration clock configured to
21 operate as the IEEE 1284 data clock line connected to data strobe 412 and the INIT
22 line 416 continues to drive the FPGA clear and reset function.

23 Data transfer continues in this manner until the FPGA 220 is fully
24 programmed by virtue of having received the correct amount of data required by the
25 particular FPGA 220 used in base station 218. Thus, each time the host software
26 is initialized, a data transfer to the FPGA 220 occurs to program the FPGA 220 to
27 function in its capacity of a virtual microcontroller (in this embodiment). Once
28 programming ceases, the FPGA 220 "wakes up" as a virtual microcontroller (or
29 whatever device is programmed into the FPGA 220 in general) and begins to
30 function as the virtual microcontroller. At this point, the interface 214 ceases to

1 function as a unidirectional programming interface and begins to function as a
2 bidirectional communication interface using the programmed operation of the
3 FPGA 220 communicating through its programmed IEEE 1248 EPP parallel
4 communication interface.

5 In the virtual microcontroller mode of operation of the FPGA 220,
6 communication is carried out using the eight data lines 420 as bidirectional data
7 lines compliant with IEEE 1284 EPP parallel communication protocol with the data
8 strobe line 412 used as a data clock and the INIT line 416 continuing to act as a
9 clear and reset signal. INIT line 416 can thus be used to reinitialize the
10 programming of the FPGA 220, for example, to revise a design parameter or to
11 simply restart the ICE system. **TABLE 2** below summarizes the significant
12 connections of this interface.

Interface Lines	Program Mode Function	Free Running "Awake" Mode Function
Data bits 0 through 7	Unidirectional data into the FPGA	Bidirectional EPP compliant communication
Data Strobe	Unidirectional programming clock	EPP Compliant Data Strobe
INIT	Low signal indicates clear configuration memory and prepare to receive new configuration data	Low signal indicates clear configuration memory and enter programming mode - prepare to receive new configuration data

18 **TABLE 2**

19
20 The programming and communication process between the host 210 and
21 the FPGA 220 is described in flow chart 500 of **FIGURE 7** starting at 502. The host
22 software is loaded and initialized at 506, and asserts a logic low on the INIT line
23 416 to signal a reset and clearing of the FPGA 220's configuration memory 424 at
24 510. In response to this signal, the FPGA 220 clears configuration memory 424 at

1 514. The Host computer 210 then begins transferring a new set of configuration
2 parameters to the FPGA 220 at 520 by strobing data into the FPGA's configuration
3 memory 424. This set of configuration parameters configures the FPGA 220 to
4 have an IEEE 1284 EPP compliant communication interface. In other
5 embodiments, other modes of communication could also be used (e.g., extended
6 communication port (ECP) or serial communications) could be used without
7 departing from the invention.

8 This process continues at 526 until all data are transferred at 530. The
9 FPGA 220 then wakes up to operate with the new configuration parameters stored
10 in configuration memory 424 at 534. The FPGA 220 continues to operate as
11 configured at 538 until such time as the INIT line 416 is again asserted by the Host
12 computer 210 at 544. Control then returns to 514 where the FPGA 220 is cleared
13 and the reprogramming process proceeds as previously described.

14 Using this mechanism, the FPGA 220 can be coupled to the host computer
15 210 using a single interface 214 for both programming the FPGA 220 and for later
16 communication with the FPGA 220 operating as the virtual microcontroller. This
17 avoids use of multiple interface connections and/or use of a separate processor to
18 handle details associated with configuration programming and communication with
19 the FPGA 220.

20 The present invention provides for full in-circuit emulation without need for
21 a special bond-out version of a DUT. This is accomplished using a minimal
22 amount of design embedded within the DUT itself. In the current embodiment, the
23 only functionality required of the production microcontroller itself is to provide for
24 transfer of data over two lines forming the data portion of the interface and reading
25 commands for break, watchdog and interrupt functions received over the same two
26 data lines. These provisions are simple to implement, and use minimal circuitry.
27 The two additional pinouts used for this function were readily accommodated in the
28 eight bit microcontroller of the current invention. Moreover, the use of a single
29 standard IEEE 1284 printer cable interface between the virtual microcontroller and
30 the host computer to provide both FPGA programming and communication

1 between the ICE system and the Host processor provides for a simple and versatile
2 implementation.

3 In the above described In-Circuit Emulator system, the virtual microcontroller
4 and the real microcontroller operate in lock-step as previously described. A
5 problem can arise in trying to keep the two microcontrollers operating in lock-step
6 synchronization when an I/O read operation is performed followed by a conditional
7 jump (or branch) based on the result of the I/O read. The problem involves keeping
8 the In-Circuit Emulator operating in lock-step operation with the real micro
9 controller.

10 As previously described, the real microcontroller 232 incorporates an I/O bus
11 and other circuitry that is not present in the virtual microcontroller 220. Thus,
12 transfers of I/O read information from the microcontroller 232 to the virtual
13 microcontroller 220 are used to provide the results of I/O reads to the virtual
14 microcontroller. Although the microcontroller 232 has access to I/O read
15 information immediately after the read, the virtual microcontroller 220 waits for data
16 to be transferred over bus 226 in order to obtain the I/O read information.

17 When an instruction following an I/O read is a conditional jump instruction
18 based on the result of the I/O read, the virtual microcontroller 220 may not have
19 enough time to properly compute the location of a conditional jump. For example,
20 consider the following two instructions that are executed in both microcontrollers.

21
22 tst io[08h], ffh
23 jz f00_label
24

25 The first instruction specifies reading an I/O address 08h and compares the
26 value there with the value in memory location ffh. If the condition of the comparison
27 is met, the "zero flag" is set. The second instruction specifies that if the zero flag
28 is set, a jump is to be made to a specified instruction. The first instruction tells the
29 microcontroller 232 to obtain I/O data from a peripheral location. At the end of the
30 test instruction the microcontroller's zero flag is set, but the I/O data has not yet

1 been forwarded to the virtual microcontroller. In fact, the I/O data will not be sent
2 to the virtual microcontroller until just before the end of the conditional jump
3 instruction. However, both the virtual microcontroller 220 and the microcontroller
4 232 need to execute the instructions and remain in lock-step synchronization.

5 In order to overcome this problem, the virtual microcontroller always
6 assumes that a jump condition is true and computes the target jump location as if
7 the condition requiring the jump has been satisfied. This permits the virtual
8 microcontroller to compute the target jump location as the I/O data is being
9 received. Just before the actual jump is performed, the virtual microcontroller 220
10 has time to evaluate the conditional jump and then, depending on the outcome of
11 the evaluation, either use the pre-computed jump information if the condition is true
12 or simply increment the program counter if the jump condition fails.

13 This process is described in connection with process 600 of **FIGURE 8**
14 starting at 604. At 608 an instruction is received and at 612 a determination is
15 made whether or not an I/O read followed by a conditional branch operation that
16 depends on the result of the I/O read has been encountered. If not, normal
17 operation proceeds at 616 and control returns to 608 to retrieve the next instruction.
18 However, if an I/O read followed by a conditional branch instruction has been
19 detected at 612, the address of the next consecutive instruction is saved at 620
20 since this is a very easy operation to perform. At 624 the virtual microcontroller 220
21 assumes that a jump is going to be required and pre-computes the address to be
22 jumped to. This computed address is then saved. When the I/O read data has
23 been completely received at 628 a determination is made at 632 whether or not an
24 actual jump to the address computed in 624 is required. If not, control passes to
25 636 where the saved address of the next consecutive instruction is retrieved at 636.
26 The next instruction is then executed at 616 and control returns to 608 where the
27 next instruction is retrieved.

28 If, however, a jump is to be performed as a result of the I/O read data at 632,
29 the address of the jump as computed in 624 is retrieved at 640 and the instruction
30 at the jump address is retrieved at 644. Control then passes to 616 where the

1 instruction retrieved at 644 is executed. Control then returns to 608 where the next
2 consecutive address following the retrieved jump location is retrieved.

3 Using this technique, speculative execution is carried out to compute the
4 potential jump address so that a jump can be rapidly made when a determination
5 based on the result of I/O read data is received at the virtual microcontroller 220.
6 Thus, even when an I/O read followed by a conditional jump is encountered, the
7 real microcontroller 232 and virtual microcontroller 220 can continue to operate in
8 lock-step.

9 While the present embodiment is implemented using a processor that does
10 not use pipelined instructions, this is not to be considered limiting. As long as
11 adequate time is available to serialize and transmit data over the interface, the
12 present interface and break management techniques could equally well be
13 implemented in a pipelined processor.

14 Those skilled in the art will understand that although the current invention
15 has been explained in terms of providing in-circuit emulation of the core processing
16 functions of a microcontroller. However, the present invention can be realized for
17 any complex electronic device for which in-circuit emulation is needed including,
18 but not limited to, microprocessors and other complex large scale integration
19 devices without limitation. Moreover, although the mechanism for use of the
20 interface between the host processor and the FPGA has been described in the
21 environment of an ICE system, this should not be considered limiting since this
22 interface mechanism can be used for other systems requiring FPGA programming
23 and communication functions over a single interface.

24 Those skilled in the art will recognize that the present invention has been
25 described in terms of exemplary embodiments based upon use of a programmed
26 processor. However, the invention should not be so limited, since the present
27 invention could be implemented using hardware component equivalents such as
28 special purpose hardware and/or dedicated processors which are equivalents to
29 the invention as described and claimed. Similarly, general purpose computers,
30 microprocessor based computers, micro-controllers, optical computers, analog